Compact and Provably Secure Lattice-Based Signatures in Hardware

James Howe, Ciara Rafferty, Ayesha Khalid, and Máire O'Neill
Centre for Secure Information Technologies (CSIT),
Queen's University Belfast, Northern Ireland,
{jhowe02, c.m.rafferty, a.khalid, maire.oneill}@qub.ac.uk

***Abstract*—Lattice-based cryptography is a quantum-safe alternative to existing classical asymmetric cryptography, such as RSA and ECC, which may be vulnerable to future attacks in the event of the creation of a viable quantum computer. The efficiency of lattice-based cryptography has improved over recent years, but there has been relatively little investigation into hardware designs of digital signature schemes. In this paper, the first hardware design of the provably secure Ring-LWE digital signature scheme, Ring-TESLA, is presented, targeting a Xilinx Spartan-6 FPGA. The results better compactness of all previous lattice-based digital signature schemes in hardware, and can achieve between 104-785 signatures and 102-776 verifications per second.**

Keywords*—*lattice-based cryptography, digital signatures, post-quantum cryptography, hardware security, FPGA.

I. Introduction

A digital signature scheme (DSS) is important for building secure systems and is widely used in most real world security protocols. Almost all currently used DSSs are based on the hardness of the factoring problem (RSA) or the discrete logarithm problem (DSA/ECDSA). With the potential advent of a quantum computer in the not too distant future, current asymmetric cryptography would be rendered insecure. Indeed, quantum computers are expected to break all currently secure instances of RSA and ECDSA in polynomial time. One potential solution is to adopt the use of DSSs based on the hardness of certain lattice problems which are assumed to be resistant to quantum attacks.

Due to significant research advancements in recent years, lattice-based schemes have now become viable alternatives to existing asymmetric cryptography. There are several practical proposals based on varying hard lattice-based problems, such as the NTRU cryptosystem and and Ring-LWE [1], [2]. For further background on practical lattice-based DSSs, the reader is referred to a recent survey on the state-of-the-art [3].

Previous hardware designs of lattice-based DSSs, the GLP scheme [4] and the BLISS scheme [5], demonstrate good performance and outperform ECDSA and RSA. However, significant hardware resources are consumed by the costly discrete Gaussian sampler in the BLISS scheme. Additionally, the schemes rely on extra security assumptions, which do not offer the very appealing average-case to worst-case hardness property offered by Ring-LWE. This quality renders *all* cryptographic constructions based on it secure, under the assumption that worst-case lattice problems are hard. Furthermore, for both GLP and BLISS, their parameters are not chosen directly from their security reduction, meaning their instantiations are not provably secure.

An alternative lattice-based signature scheme, Ring-TESLA [2], has been proposed which does not require discrete Gaussian sampling during sign or verify, and offers average-case to worst-case hardness with a tight security reduction and a provably secure instantiation. A tight security reduction implies the cryptoscheme is no easier to solve than its hardness problem. Ring-TESLA competes well with GLP and BLISS in software [2], but as yet no hardware designs exist.

This paper presents the first hardware designs of the Ring-TESLA signature scheme. The proposed designs are compact, targeting long-term security and low-area applications. The paper is structured as follows: the Ring-TESLA scheme is detailed in Section II. Section III outlines the proposed hardware designs of the signature scheme and results are given in Section IV.

II. Ideal Lattice-Based Signatures

Lattice-based cryptography is emerging as a promising quantum-resistant alternative to ECC or RSA, and offers efficient performance for both encryption and signatures. For significant efficiency gains, ideal lattices are usually used which allow for smaller key sizes and faster computations by computing over a specific algebraic structure. The Ring-LWE problem [6], commonly used in lattice-based cryptography, is well studied and demonstrates strong computational hardness.

The most practical lattice-based DSSs are based upon the Fiat-Shamir paradigm [3], such as the state-of-the-art BLISS by Ducas *et al.* [1], which is based upon ideal lattices with NTRU assumptions. NTRU cryptoschemes have existed for a significant period of time, with the only current serious break in NTRU-based schemes targeted NTRUSign [7]. However, the hardness assumptions of NTRU is not related to the hardness of worst-case lattice problems, a useful property of Ring-LWE [8]. Accordingly, a lattice-based DSS based on the Ring-LWE problem has been proposed by Akleylek *et al.* [2], named Ring-TESLA. Ring-TESLA provides three appealing properties.

Firstly, Ring-TESLA provides a tight security reduction, a provably secure instantiation, and worst-case hardness, which is not provided by GLP or BLISS. Cryptoschemes that have provably secure instantiations are considered stronger in the sense of security [9], especially when non-tight cryptoschemes have been shown to provide weaker security assurances [10].

Secondly, the Ring-TESLA Sign and Verify algorithms do not require discrete Gaussian sampling, which is instead

Name of the scheme	Parameters
Security	128-bits
Lattice dimension n	512
Modulus q, $(\log_2(q))$	51750913,(26)
Weight of the challenge ω	19
Gaussian std. dev. σ	52
Dropped bits d	23
Error threshold L	2766
Sign/Verify thresholds B, U	$2^{22}-1, 3173$
Repetition rate	2.9

TABLE I: The 128-bit parameter set for Ring-TESLA.

computed during key generation, off-device. Discrete Gaussian samplers are known to be susceptible to side-channel analysis [11], [12], hence avoiding this on-device is advantageous. Additionally, this also economises the overall hardware resources used, for example, the discrete Gaussian sampling module consumes $\approx 15\%$ of the overall resources in for BLISS hardware design [5].

Thirdly, Ring-TESLA competes well in terms of throughput, with the other lattice-based signatures GLP and BLISS in software, despite having larger input and output sizes [2].

The parameter set (Table I) provides 128-bit security. The modulus is increased from $q = 39960577$ to $q = 51750913$, as the security reduction with the original modulus caused a slightly bigger security gap than expected[1]. The Ring-TESLA schemes Sign_{sk} and Verify_{pk} are outlined in Algorithms 1 and 2, with $\text{KeyGen}(1^n)$ carried out offline. The required hardware modules for Sign and Verify are polynomial multiplication over $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$, addition/subtraction, modular reduction, and hashing.

Algorithm 1 Signing Algorithm for Ring-TESLA

procedure SIGN(μ, $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{s}$, $\mathbf{e}_1$, $\mathbf{e}_2$)
  $\mathbf{y} \xleftarrow{\$} \mathcal{R}_{q,[B]}$
  $\mathbf{v}_1 \equiv \mathbf{a}_1\mathbf{y} \mod q$, $\mathbf{v}_2 \equiv \mathbf{a}_2\mathbf{y} \mod q$
  $c = H(\lfloor \mathbf{v}_1 \rceil_{d,q}, \lfloor \mathbf{v}_2 \rceil_{d,q}, \mu)$
  $\mathbf{c} = F(c)$
  $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{sc}$
  $\mathbf{w}_1 \equiv \mathbf{v}_1 - \mathbf{e}_1\mathbf{c} \mod q$, $\mathbf{w}_2 \equiv \mathbf{v}_2 - \mathbf{e}_2\mathbf{c} \mod q$
  if $[\mathbf{w}_1]_{2^d}, [\mathbf{w}_2]_{2^d} \notin \mathcal{R}_{2^d-L}$ **or** $\mathbf{z} \notin \mathcal{R}_{B-U}$ **then**
    Restart
  return $(\mathbf{z}, c)$

Algorithm 2 Verification algorithm for Ring-TESLA

procedure VERIFY(μ, $\mathbf{z}$, c, $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{t}_1$, $\mathbf{t}_2$)
  $\mathbf{c} = F(c)$
  $\mathbf{w}'_1 \equiv \mathbf{a}_1\mathbf{z} - \mathbf{t}_1\mathbf{c} \mod q$, $\mathbf{w}'_2 \equiv \mathbf{a}_2\mathbf{z} - \mathbf{t}_2\mathbf{c} \mod q$
  $c' = H(\lfloor \mathbf{w}'_1 \rceil_{d,q}, \lfloor \mathbf{w}'_2 \rceil_{d,q}, \mu)$
  if $c = c'$ **and** $\mathbf{z} \in \mathcal{R}_{B-U}$ **then**
    return 1
  else
    return 0

Key generation firstly generates $\mathbf{s}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow D^n_\sigma$, with $\mathbf{e}_1$ and $\mathbf{e}_2$ checked for validity [2]. Two Ring-LWE polynomials are then calculated $\mathbf{t}_1 \equiv \mathbf{a}_1\mathbf{s} + \mathbf{e}_1 \mod q$ and $\mathbf{t}_2 \equiv \mathbf{a}_2\mathbf{s} + \mathbf{e}_2 \mod q$ as the scheme's public-key (pk), with the polynomials $\mathbf{s}, \mathbf{e}_1, \mathbf{e}_2$ being the scheme's secret-key (sk).

[1]The updated modulus is found on the Ring-TESLA homepage (https://tesla.informatik.tu-darmstadt.de/de/tesla/) and was verified with the authors.

To sign a message μ, a uniform polynomial $\mathbf{y} \xleftarrow{\$} \mathcal{R}_{q,[B]}$ is generated for use in the calculation of the signature and for validity checks. Firstly, it is used to calculate intermediate polynomials $\mathbf{v}_1 \equiv \mathbf{a}_1\mathbf{y} \mod q$ and $\mathbf{v}_2 \equiv \mathbf{a}_2\mathbf{y} \mod q$, which are input into the hash function $H(\cdot)$ along with the message data μ to output the bit-string c. An encoding function $F : \{0,1\}^\kappa \rightarrow \mathbb{B}_{n,\omega}$ (as described in [13]) then maps this bit-string to a LHW polynomial $\mathbf{c}$, with ω 1 values and $n-\omega$ 0 values. The LHW polynomial $\mathbf{c}$ is then used to calculate the signature $\mathbf{z} \equiv \mathbf{y} + \mathbf{sc}$ as well as the polynomials $\mathbf{w}_1 \equiv \mathbf{v}_1 - \mathbf{e}_1\mathbf{c} \mod q$ and $\mathbf{w}_2 \equiv \mathbf{v}_2 - \mathbf{e}_2\mathbf{c} \mod q$ which are used to check the validity of a signature. The verification algorithm is essentially equivalent to the signing algorithm without uniform polynomial generation and the calculation of $\mathbf{z}$.

III. Hardware Implementation

Two separate hardware designs for the Sign and Verify algorithms in the Ring-TESLA DSS are proposed. The required modules for both Sign and Verify are a polynomial multiplier, a hash function and a low Hamming weight (LHW) multiplier. Figure 1 illustrates the proposed hardware design of Ring-TESLA Sign. The Verify design is adapted from the proposed Sign hardware design.

A. Hardware components

Polynomial multiplication is the most expensive module required in the proposed designs in terms of latency consumption. The most commonly chosen method for modular polynomial multiplication is the number theoretic transform (NTT), which offers fast performance and incorporates the reduction modulo q. However, it is costly in terms of hardware resource usage and there are significant restrictions on the parameter selection when a NTT multiplier is used. Alternatively, traditional multiplication techniques can be employed to carry out the polynomial multiplication operations, incurring an additional latency cost. An additional modular reduction module is therefore required.

In this design, the polynomial multiplication is carried out using a variant of schoolbook multiplication, known as Comba multiplication [14], which improves the performance by combining carry handling and reducing write access to memory. This multiplier type fits within the overall compact design goal, unlike NTT. The Comba multiplier is particularly suitable for the FPGA platform and exploits the fast arithmetic within the DSP units [15]. The multiply-and-accumulate (MAC) operations are computed within each DSP slice until the inner products of the schoolbook multiplier are complete.

The modular reduction component uses the Barrett method [16]. Any generic modulus can be used within Barrett reduction, with only one pre-computation is required. Two multiplication units with a maximum of two subtractions are needed in Barrett reduction [17]. In the proposed designs, an individual Comba multiplier is reused to carry out the modular reduction. The combination of Comba multiplication and Barrett reduction create an overall modular multiplication component, which multiplies polynomials over the ring $\mathcal{R}_q$.

The SHA3 hash function, Keccak, is chosen as the random oracle in the proposed Ring-TESLA designs, due to its speed in hardware as well as its post-quantum security [18]. A LHW

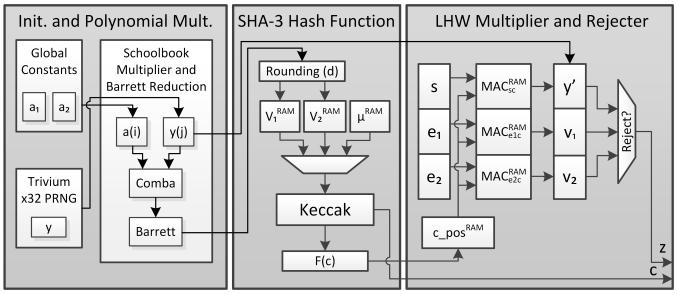

Fig. 1: A block diagram of the proposed hardware design for Ring-TESLA Sign SB-I, widths being 26-bits. The global parameters $\mathbf{a}_1$ and $\mathbf{a}_2$ are stored in BRAM18.

polynomial multiplication module is also designed to compute on the LHW output of the hash function. The LHW multiplier, used in both Sign and Verify, uses the column-wise schoolbook technique, since the LHW calculations only require a small amount of shift and adds. Column-wise is preferred over row-wise as it requires less storage.

B. Signing and Verifying Hardware Designs

Figure 1 shows the Sign hardware architecture for Ring-TESLA. Two dual-port 18 Kb block RAMs (BRAM18) are used to store the global constants $\mathbf{a}_1, \mathbf{a}_2$. An unrolled x32 Trivium component is used to generate uniform random bits for the polynomial $\mathbf{y} \xleftarrow{\$} \mathcal{R}_{q,[B]}$, whose input, as well as the input of the polynomial $\mathbf{a}_1$ and $\mathbf{a}_2$ is controlled by two counters, which increment on the ready signal of the modular multiplication component. The global constant polynomial selection ($\mathbf{a}_1$ or $\mathbf{a}_2$) is controlled by the finite state machine. Once each element of $\mathbf{v}_1$ and $\mathbf{v}_2$ is output from the modular multiplication module, the d least significant bits of each are stored in RAM for use in the hash function, where the full values are also stored for use in the rejection stage.

The binary string output from the hash-function (c) is input to the encoding function $F(c)$, which outputs a LHW polynomial $\mathbf{c}$. This prior knowledge of a LHW polynomial allows the use of a LHW polynomial multiplier, which only computes values for non-zero elements. The discrete Gaussian distributed (D_σ) secret-keys $\mathbf{s}, \mathbf{e}_1, \mathbf{e}_2$ are also LHW since, for instance, the probability a sample $x \leftarrow D_\sigma$ also satisfies $x \in \{-100, 100\}$ (thus 6-7 bits in length) is around 95%.

The LHW polynomial $\mathbf{c}$ is an input into every LHW computation. This includes the calculation of the ciphertext $\mathbf{z}$, and the variables used to accept or reject the signature, $\mathbf{w}_1$ and $\mathbf{w}_2$. The LHW polynomial multiplier exploits the fact that the hash output polynomial $\mathbf{c}$ has only $\omega = 19$ elements equal to a one, and $n-\omega = 493$ zero elements. The same LHW multiplier component is reused to calculate the polynomials $\mathbf{z}$, $\mathbf{w}_1$, and $\mathbf{w}_2$, sequentially. Once a coefficient is output from the LHW multiplier, it is processed by the rejecter, which checks whether its size is valid for output. Rejecting a signature element-wise is preferable to minimise run-time.

The column-wise LHW module stores the secret-key values ($\mathbf{s}$, $\mathbf{e}_1$, and $\mathbf{e}_2$) in a single-port distributed RAM, and takes the input $\mathbf{c}$ from the hash function. Then, for each element in $\mathbf{s}$, $\mathbf{e}_1$, and $\mathbf{e}_2$, the multiplier accumulates the inner products using a MAC unit. Once each column-wise element is calculated, it is added/subtracted to/from the corresponding values in $\mathbf{v}_1$, $\mathbf{v}_2$, or $\mathbf{y}'$, and the final values are checked with their rejection conditions (Line 8 in Algorithm 1), where only the signature $\mathbf{z}$ and hash-string c are stored. This approach is advantageous since the rejection validity is calculated instantly and in parallel to the next LHW element calculation.

The Sign and Verify FSM operates in two-stages and in a pipelined fashion, due to the latency of the calculations of $\mathbf{v}_1$ and $\mathbf{v}_2$. For the first signature, $\mathbf{y}$ is generated during initialisation when the global constants are read in, with an additional $\mathbf{y}'$ polynomial (see Figure 1) generated during the calculation of $\mathbf{v}_1$ and $\mathbf{v}_2$. Once these calculations have finished, $\mathbf{y}$ is swapped with $\mathbf{y}'$, so the calculations of $\mathbf{v}_1$ and $\mathbf{v}_2$ for the next signature can begin again. This removes the hashing and LHW calculations from the critical path and cycle count, as well as savings for generating a new polynomial $\mathbf{y}$.

Verify operates in a similar fashion to Sign. The Comba/Barrett polynomial multiplier calculates $\mathbf{a}_1\mathbf{z}$ and $\mathbf{a}_2\mathbf{z}$, with $\mathbf{z}$ being stored in BRAM. The results are also stored in RAM and are updated after subtraction with the LHW calculations of $\mathbf{t}_1\mathbf{c}$ and $\mathbf{t}_2\mathbf{c}$. The final results are input into the hash function, where the signature is validated if the hash value matches the hash-string input from the signature.

C. Optimised design for accelerated performance

There is a trade-off between the latency achievable by the proposed Ring-TESLA hardware designs of Sign and Verify and the associated hardware resource usage. The proposed designs, named SB-I Sign and SB-I Verify, offer reduced area consumption at the cost of additional latency. In these designs, a standard Comba multiplier with one Barrett modular reduction unit is employed. The Barrett modular reduction unit is carried out in parallel while the Comba multiplier is used to minimise latency of the modular multiplication unit. These designs target low area applications, where there is a need for a provably secure instantiation and/or reduced area consumption, and where a slower performance may be acceptable.

The bottleneck in Ring-TESLA is polynomial multiplication. Three additional designs SB-II, SB-IV, and SB-VIII, are undertaken having two, four, and eight parallel Comba multipliers, respectively. These parallelised designs utilise extra BRAM for $\mathbf{a}_1$ and $\mathbf{a}_2$ access. To minimise latency, in all proposed designs only one modular reduction is employed, running in parallel with the multiple Comba multiplier units.

The modulus in this scheme has a length of $\log_2(q) =$ 26-bits. For each 26-bit multiplication, the Comba multiplier occupies 2 DSP slices on the target Spartan-6 FPGA.

IV. Results and Conclusions

The proposed architectures are implemented using the Xilinx ISE Design Suite 14.7 synthesis tool. The target device is a Xilinx Spartan-6 FPGA (S6 LX25). Table II shows the post-place and route results for the proposed hardware designs of Ring-TESLA Sign and Verify. These designs fit comfortably on the low-end FPGA. As expected, the optimised designs of SB-II, SB-IV, and SB-VIII have reduced latency in comparison to SB-I, at the cost of additional area consumption. Results indicate that up to 785 operations per second can be achieved by the proposed designs, with an associated low area cost.

TABLE II: Post-place and route results of the proposed hardware designs for Ring-TESLA [2], with a summary of other DSSs, including GLP-I [4], BLISS-I [5], RSA, and ECDSA (results taken from [3]).

Operation, Configuration	Security	Device	LUT/FF/Slice	BRAM/DSP	MHz	Cycles	Ops/s
Ring-TESLA (Sign, SB-I)	128-bits	S6 LX25	4447/3345/1257	3/6	190	1835540	104
Ring-TESLA (Sign, SB-II)	128-bits	S6 LX25	4828/3790/1513	4/8	196	917771	214
Ring-TESLA (Sign, SB-IV)	128-bits	S6 LX25	5071/3851/1503	4/12	187	458891	408
Ring-TESLA (Sign, SB-VIII)	128-bits	S6 LX25	6848/5457/2254	4/16	180	229446	785
Ring-TESLA (Verify, SB-I)	128-bits	S6 LX25	3714/3023/1172	3/6	188	1835540	102
Ring-TESLA (Verify, SB-II)	128-bits	S6 LX25	3917/3253/1238	3/8	194	917771	212
Ring-TESLA (Verify, SB-IV)	128-bits	S6 LX25	4793/3939/1551	3/12	186	458891	406
Ring-TESLA (Verify, SB-VIII)	128-bits	S6 LX25	6473/5582/2103	3/16	178	229446	776
GLP (Sign, Schoolbook x3)	80-bits	S6 LX16	7465/8993/2273	30/28	162	-	931
GLP (Verify, Schoolbook x3)	80-bits	S6 LX16	6225/6663/2263	15/8	158	-	998
BLISS (Sign, NTT)	128-bits	S6 LX25	7193/6420/2291	6/5	139	15864	8761
BLISS (Verify, NTT)	128-bits	S6 LX25	5065/4312/1687	4/3	166	16346	17101
RSA (Sign)	103-bits	V5 LX30	3237 slices	7/17	200	-	89
ECDSA (Sign)	128-bits	V5 LX110	32299 LUT/FF pairs	10/37	139	-	-
ECDSA (Verify)	128-bits	V5 LX110	32299 LUT/FF pairs	10/37	110	-	-

The proposed designs are compared to existing classical DSSs currently used in practice (RSA/ECDSA), and similar lattice-based cryptosystems (GLP/BLISS), as seen in Table II. The Ring-TESLA results compare well with or better than the classical hardware designs of RSA and ECDSA, in terms of lower area consumption and operations per second.

The proposed designs significantly better existing lattice-based DSSs in terms of area consumption. Ring-TESLA Sign SB-I results show $\approx 45\%$ FPGA slice reduction compared to GLP and BLISS, with results for SB-II and SB-IV reducing FPGA slice consumption to $\approx 34\%$. However, the results have an increase in latency compared to GLP and BLISS. The Ring-TESLA algorithms contribute to this comparative increase in latency. More specifically, Ring-TESLA Sign requires two full polynomial multiplications (for $\mathbf{v}_1$ and $\mathbf{v}_2$), whereas at the same stage GLP and BLISS only require one. The designs are also compact when considering the larger operand size for Ring-TESLA (26-bits), compared to GLP (23-bits) and BLISS (14-bits).

To conclude, the first hardware designs of the provably secure Ring-TESLA signature scheme are proposed in this research, targeting a low-cost Spartan-6 FPGA device. In the context of design space exploration, using multiple parallel Comba multipliers (plus a Barret modular reduction) instead of an NTT was chosen for design compactness. The designs fit comfortably on a low-end Spartan-6 FPGA, despite having large memory requirements and computationally expensive algorithms. It understandably lacks in throughput performance compared to other lattice-based DSSs, but this is the trade-off for the stronger security it provides. The proposed hardware designs are practical for low-area or high security applications, where tighter security reductions are desirable, at the cost of a slower performance.

REFERENCES

[1] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal Gaussians," in *CRYPTO (1)*, 2013, pp. 40–56, full version: https://eprint.iacr.org/2013/383.pdf.

[2] S. Akleylek, N. Bindel, J. A. Buchmann, J. Krämer, and G. A. Marson, "An efficient lattice-based signature scheme with provably secure instantiation," in *AFRICACRYPT*, 2016, pp. 44–60.

[3] J. Howe, T. Pöppelmann, M. O'Neill, E. O'Sullivan, and T. Güneysu, "Practical lattice-based digital signature schemes," *ACM TECS*, vol. 14, no. 3, p. 41, 2015.

[4] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," in *CHES*, 2012, pp. 530–547.

[5] T. Pöppelmann, L. Ducas, and T. Güneysu, "Enhanced lattice-based signatures on reconfigurable hardware," in *CHES*, 2014, pp. 353–370, full version: https://eprint.iacr.org/2014/254.pdf.

[6] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM*, vol. 60, no. 6, p. 43, 2013.

[7] L. Ducas and P. Q. Nguyen, "Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures," in *ASIACRYPT*, 2012, pp. 433–450.

[8] D. Stehlé and R. Steinfeld, "Making NTRU as secure as worst-case problems over ideal lattices," in *EUROCRYPT*. Springer, 2011, pp. 27–47.

[9] M. Bellare and P. Rogaway, "The exact security of digital signatures-how to sign with RSA and Rabin," in *EUROCRYPT*. Springer, 1996, pp. 399–416.

[10] S. Chatterjee, A. Menezes, and P. Sarkar, "Another look at tightness," in *Selected Areas in Cryptography*. Springer-Verlag, 2011, pp. 293–319.

[11] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, Gauss, and reload – a cache attack on the BLISS lattice-based signature scheme," Cryptology ePrint Archive, Report 2016/300, 2016.

[12] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill, "On Practical Discrete Gaussian Samplers For Lattice-Based Cryptography," *IEEE Transactions on Computers*, 2016.

[13] S. Bai and S. D. Galbraith, "An improved compression technique for signatures based on learning with errors," in *CT-RSA*, 2014, pp. 28–47.

[14] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM systems journal*, vol. 29, no. 4, pp. 526–538, 1990.

[15] T. Güneysu, "Utilizing hard cores of modern FPGA devices for high-performance cryptography," *J. Cryptographic Engineering*, vol. 1, no. 1, pp. 37–55, 2011.

[16] P. Barrett, "Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *CRYPTO*, 1986, pp. 311–323.

[17] J. F. Dhem, "Design of an efficient public-key cryptographic library for RISC-based smart cards," Ph.D. dissertation.

[18] M. Amy, O. D. Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck, "Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3," in *SAC*, 2016.